

# **Exhibit D**

Claim element	Accused Instrumentalities
8a. A method of providing data to a data requester comprising:	<p>Apple has committed and continues to commit acts of infringement under 35 U.S.C. § 271 (i) with any version of iTunes that can access the iTunes Store; (ii) with any version of the App Store app that can access the App Store app; (iii) with any version of any Apple iOS hardware or software product (e.g., including Apple’s various iPhone products (e.g., iPhone, iPhone 3G, iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPhone 5C, iPhone 5S, and later), Apple’s various iPad products (e.g., iPad, iPad 2, iPad 3rd Generation, iPad 4th Generation, iPad Mini, iPad Mini 2nd Generation, iPad Air, and later), Apple’s various iPod Touch products (iPod Touch, iPod Touch 2nd Generation, iPod Touch 3rd Generation, iPod Touch 4th Generation, iPod Touch 5th Generation, and later) Apple’s various operating system softwares, etc.) that includes any version of the iTunes Store or the App Store app that can access the iTunes Store or the App Store; (iv) with any version of the Mac App Store; (v) with any version of any Apple Mac OS hardware or software product (e.g., including Apple’s various Mac computer products (e.g., MacBook Pro, MacBook Air, Mac mini, Mac Pro, iMac, and later), Apple’s various operating system software, etc.) that includes any version of the iTunes Store or the Mac App Store app that can access the iTunes Store or Mac App Store; (vi) with any version of any Apple TV hardware or software products (e.g., including Apple TV, Apple TV 2nd Generation, Apple TV 3rd Generation, and later, and modified builds of Mac OS or iOS software, such as Apple TV Software); and (vii) with Apple’s internal servers involved in operating Apple’s iTunes Store, Apple’s App Store, Apple’s Mac App Store, Apple’s in-application payment functionality, and Apple’s iAd Network (collectively referred to as “Apple’s Accused Instrumentalities”).</p> <p>Robot Entertainment has committed and continues to commit acts of infringement under 35 U.S.C. § 271 with its “Hero Academy” app.</p> <p>KingsIsle has committed and continues to commit acts of infringement under 35 U.S.C. § 271 with its “Grub Guardian” and “WizardBlox” apps.</p> <p>Game Circus has committed and continues to commit acts of infringement under 35 U.S.C. § 271 with its apps, including its apps that use Apple’s in-application payment functionality to collect payment for enhanced functionality or additional content and with its apps that provide in-application advertising.</p> <p>Apple’s Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise an infringing method of providing data to a data requester.<sup>1</sup> See, e.g.,:</p>

<sup>1</sup> See Apple, Inc., *Apple – iPhone 5s – Built-in Apps* (<http://www.apple.com/iphone-5s/built-in-apps/>) [hereinafter “Built-in Apps”]; see Apple, Inc., *Apple – iPhone 5s – Features* (<http://www.apple.com/iphone-5s/features/>) [hereinafter “Features”]; see Apple, Inc., *Apple – iPhone 5s – Technical Specifications* (<http://www.apple.com/iphone-5s/specs/>) [hereinafter “Specifications”]; Apple, Inc., *Hero Academy for iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPod touch (3rd generation), iPod touch (4th generation), iPod touch (5th*

<p>8b. receiving a request for a data item from the requester;</p>	<p>Apple's Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise the step of receiving a request for a data item from the requester. <i>See, e.g.,</i>:</p> <p>"SKProduct objects are returned as part of an SKProductsResponse object. Each product object provides information about a product you previously registered in iTunes Connect."</p> <p><b>Source:</b> Store Kit at 38.</p> <p>"An SKProductsRequest object is used to retrieve localized information about a list of products from the Apple App Store. Your application uses this request to present localized prices and other information to the user without having to maintain that list itself.</p> <p>To use an SKProductsRequest object, you initialize it with a list of product identifier strings, attach a delegate, and then call the request's start (page 52) method. When the request completes, your delegate receives an SKProductsResponse object."</p> <p><b>Source:</b> Store Kit at 43; <i>see also</i> Store Kit at 43-44.</p> <p>"SKRequest is an abstract class representing a request to the Apple App Store. Subclasses of SKRequest represent different kinds of requests.</p> <p>To use a request object, initialize a subclass of SKRequest and set the delegate (page 51) property, then call the start (page 52) method."</p>
--	--

generation) and iPad on the iTunes App Store (<https://itunes.apple.com/us/app/hero-academy/id488156323?mt=8>) [hereinafter "Hero Academy iTunes"]; Apple, Inc., *Grub Guardian for iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPod touch (3rd generation), iPod touch (4th generation), iPod touch (5th generation) and iPad on the iTunes App Store* (<https://itunes.apple.com/us/app/grub-guardian/id498330099?mt=8>) [hereinafter "Grub Guardian iTunes"]; Apple, Inc., *Prize Claw for iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPod touch (3rd generation), iPod touch (4th generation), iPod touch (5th generation) and iPad on the iTunes App Store* (<https://itunes.apple.com/us/app/prize-claw/id434800417?mt=8&ign-mpt=uo%3D2>) [hereinafter "Prize Claw iTunes"]; iFixit, *iPhone 5s Teardown – Page 2 – iFixit* (<http://www.ifixit.com/Teardown/iPhone+5s+Teardown/17383/2>) [hereinafter "Teardown Page 2"]; Chipworks, Inc., *Inside the iPhone 5s* (<http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/inside-the-iphone-5s/>) [hereinafter "Inside iPhone 5s"]; Apple, Inc., *iPhone User Guide for iOS 7 Software* (2013) [hereinafter "iPhone User Guide"]; Apple, Inc., *In-App Purchase Programming Guide* (Sep. 2012) [hereinafter "In-App Purchases 2012"]; Apple, Inc., *Store Kit Framework Reference* (Sept. 2013) [hereinafter "Store Kit"]; Apple, Inc., *In-App Purchase Programming Guide* (Sep. 2013) [hereinafter "In-App Purchases 2013"]; Apple, Inc., *Getting Started with In-App Purchase on iOS and OS X* (Sep. 2013) [hereinafter "Getting Started"]; Apple, Inc., *iTunes Store – Terms and Conditions*, ([www.apple.com/legal/internet-services/itunes/us/terms.html](http://www.apple.com/legal/internet-services/itunes/us/terms.html)) [hereinafter "iTunes Terms"]; Apple, Inc., *iTunes Connect Developer Guide* (Aug. 2013) [hereinafter "Connect"].

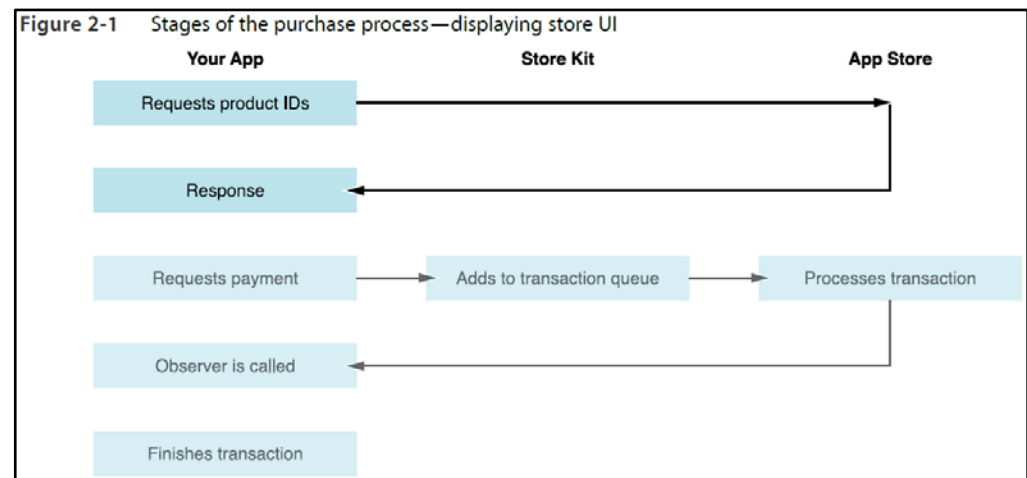
**Source:** Store Kit at 50; *see also* Store Kit at 50-52.

“A `SKStoreProductViewController` object presents a store that allows the user to purchase other media from the App Store. For example, your app might display the store to allow the user to purchase another app.

To display a store, create a new `SKStoreProductViewController` object and set its delegate. Then, present the view controller modally from another view controller in your app. Your delegate dismisses the view controller when the user completes the purchase.

To choose a specific product, call the `loadProductWithParameters:completionBlock:` (page 54) method, passing the iTunes item identifier for the item you want to sell.”

**Source:** Store Kit at 53; *see also* Store Kit at 53-55.



**Source:** In-App Purchases 2013 at 11 (showing product request submission to App Store).

8c. receiving payment data from the requester relating to payment for the requested data;

Apple’s Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise the step of receiving payment data from the requester relating to payment for the requested data. *See, e.g.,*:

“The `SKMutablePayment` class defines a request to the Apple App Store to process payment for additional functionality offered by your application. A payment encapsulates a string that identifies a particular product and the

quantity of that item the user would like to purchase.

When a mutable payment is added to the payment queue, the payment queue copies the contents into an immutable request before queueing the request. Your application can safely change the contents of the mutable payment object.”

**Source:** Store Kit at 14; *see also* Store Kit at 14-16.

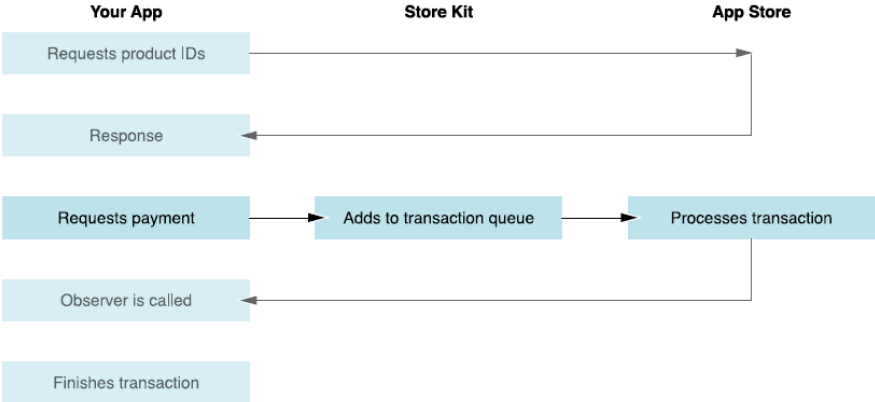
“The SKPayment class defines a request to the Apple App Store to process payment for additional functionality offered by your application. A payment encapsulates a string that identifies a particular product and the quantity of those items the user would like to purchase.”

**Source:** Store Kit at 17; *see also* Store Kit at 17-21.

“The SKPaymentQueue class provides a queue of payment transactions to be processed by the App Store. The payment queue communicates with the App Store and presents a user interface so that the user can authorize payment. The contents of the queue are persistent between launches of your app.

To process a payment, first attach at least one observer object to the queue. Then, add a payment object for the item the user wants to purchase. Each time you add a payment object, the queue creates a transaction object to process that payment and enqueues it to be processed. After payment is fulfilled, the queue updates the transaction object and then calls any observer objects to provide them the updated transaction. Your observer should process the transaction and then remove it from the queue.”

**Source:** Store Kit at 22; *see also* Store Kit at 23-31.

	<p>In the second part of the purchase process, after the user has chosen to purchase a particular product, your app submits payment request to the App Store, as shown in <a href="#">Figure 3-1</a> (page 17).</p> <p><b>Figure 3-1</b> Stages of the purchase process—requesting payment</p>  <pre> sequenceDiagram     participant YourApp as Your App     participant StoreKit as Store Kit     participant AppStore as App Store      YourApp-&gt;&gt;AppStore: Requests product IDs     AppStore--&gt;&gt;YourApp: Response     YourApp-&gt;&gt;StoreKit: Requests payment     StoreKit-&gt;&gt;StoreKit: Adds to transaction queue     StoreKit-&gt;&gt;AppStore: Processes transaction     AppStore--&gt;&gt;StoreKit: Observer is called     StoreKit-&gt;&gt;YourApp: Finishes transaction   </pre> <p><b>Source:</b> In-App Purchases 2013 at 17 (showing payment submission to App Store).</p>
<p>8d. reading the requested data from a content provider responsive to the received payment data; and</p>	<p>Apple’s Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise the step of reading the requested data from a content provider responsive to the received payment data. <i>See, e.g.,</i>:</p> <p>“The SKPaymentQueue class provides a queue of payment transactions to be processed by the App Store. The payment queue communicates with the App Store and presents a user interface so that the user can authorize payment. The contents of the queue are persistent between launches of your app.</p> <p>To process a payment, first attach at least one observer object to the queue. Then, add a payment object for the item the user wants to purchase. Each time you add a payment object, the queue creates a transaction object to process that payment and enqueues it to be processed. After payment is fulfilled, the queue updates the transaction object and then calls any observer objects to provide them the updated transaction. Your observer should process the transaction and then remove it from the queue.</p> <p>The exact mechanism you use to process a processed transaction depends on the design of your app and the product being purchased. Here are a few common examples:</p> <ul style="list-style-type: none"> <li>● If the product is a feature already built into your app, then to process the transaction, your app would enable the feature.</li> <li>● If the product includes downloadable content provided by the App Store, your app would retrieve the SKDownload objects from the transaction and ask the payment queue to download them. You would provide the actual content</li> </ul>

files to be served by the App Store to iTunes Connect when you create the product information.

● If the product represents downloadable content provided by your own server, your app might open a network connection to your server and download the content from there.”

**Source:** Store Kit at 22; *see also* Store Kit at 23-31.

“The SKPaymentTransaction class defines objects residing in the payment queue. A payment transaction is created whenever a payment is added to the payment queue. Transactions are delivered to your application when the App Store has finished processing the payment. Completed transactions provide a receipt and transaction identifier that your application can use to save a permanent record of the processed payment.”

**Source:** Store Kit at 32; *see also* Store Kit at 32-37.

“The SKReceiptRefreshRequest class allows an app to refresh its receipt. With this API, the app can request a new receipt if the receipt is invalid or missing. In the sandbox environment, you can request a receipt with any combination of properties to test the state transitions related to Volume Purchase Plan receipts.”

**Source:** Store Kit at 47; *see also* Store Kit at 47-49.

“The SKPaymentTransactionObserver protocol declares methods that are implemented by observers of an SKPaymentQueue object.

An observer is called when transactions are updated by the queue or removed from the queue. An observer should process all successful transactions, unlock the functionality purchased by the user, and then finish the transaction by calling the payment queue’s finishTransaction: (page 27) method.”

**Source:** Store Kit at 57; *see also* Store Kit at 57-61.

### “Downloading Hosted Content from Apple’s Server

**Note:** For information about associating Apple-hosted content with a product, see “Creating Products in iTunes Connect” (page 8).

When the user purchases a product that has associated hosted content, the transaction passed to your transaction queue observer also includes an instance of SKDownload which lets you download the content.

To download the hosted content, add the download objects from the transaction's downloads property to the transaction queue by calling the `startDownloads:` method of `SKPaymentQueue`. If the value of the downloads property is nil, there is no hosted content for that transaction. Unlike downloading apps, downloading hosted content does not automatically require a Wifi connection for content larger than a certain size. Avoid using cellular networks to download large files without an explicit action from user.

Implement the `paymentQueue:updatedDownloads:` method on the transaction queue observer to respond to changes in a download's state—for example, by updating progress in your UI or by notifying the user of a failed download.

Update your user interface while the content is downloading using the values of the progress and `timeRemaining` properties. You can use the `pauseDownloads:`, `resumeDownloads:`, and `cancelDownloads:` methods of `SKPaymentQueue` from your UI to let the user control in-progress downloads. Use the `downloadState` property to determine whether the download has completed. Don't use the progress or time remaining, these properties wouldn't let your app accurately tell the difference between downloads that are almost finished and downloads that are completely finished.

After the download finishes, use its `contentURL` property to locate the content.”

**Source:** In-App Purchases 2013 at 28.

#### **“Downloading Content from Your Own Server**

As with all other interactions between your app and your server, the details and mechanics of this process are up to you. The communication consists of, at minimum, the following steps:

1. Your app sends the receipt to your server and requests the content.
2. Your server validates the receipt to establish that the content has been purchased, as described in Receipt Validation Programming Guide.
3. Assuming the receipt is valid, your server responds to your app with the content.

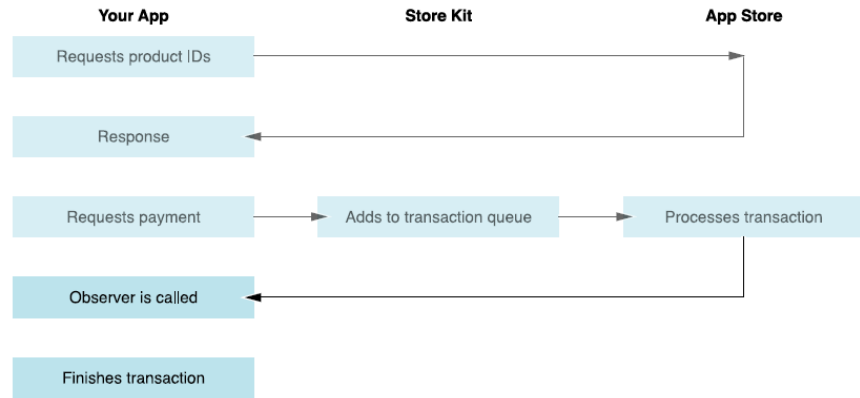
Consider the security implications of how you host your content and of how your app communicates with your server. For more information, see Security Overview.”

**Source:** In-App Purchases 2013 at 29.



In the last part of the purchase process, after the App Store has processed the payment request, your app stores information about the purchase for future launches, downloads the purchased content, and marks the transaction as finished, as shown in Figure 4-1 (page 21).

**Figure 4-1** Stages of the purchase process—delivering products



**Source:** In-App Purchases 2013 at 21 (showing transaction completion and content delivery).

8e. transmitting the read data to the requester.

Apple's Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise the step of transmitting the read data to the requester. *See, e.g.,*:

“The SKPaymentQueue class provides a queue of payment transactions to be processed by the App Store. The payment queue communicates with the App Store and presents a user interface so that the user can authorize payment. The contents of the queue are persistent between launches of your app.

To process a payment, first attach at least one observer object to the queue. Then, add a payment object for the item the user wants to purchase. Each time you add a payment object, the queue creates a transaction object to process that payment and enqueues it to be processed. After payment is fulfilled, the queue updates the transaction object and then calls any observer objects to provide them the updated transaction. Your observer should process the transaction and then remove it from the queue.

The exact mechanism you use to process a processed transaction depends on the design of your app and the product being purchased. Here are a few common examples:

- If the product is a feature already built into your app, then to process the transaction, your app would enable the feature.
- If the product includes downloadable content provided by the App Store,

your app would retrieve the SKDownload objects from the transaction and ask the payment queue to download them. You would provide the actual content files to be served by the App Store to iTunes Connect when you create the product information.

● If the product represents downloadable content provided by your own server, your app might open a network connection to your server and download the content from there.”

**Source:** Store Kit at 22; *see also* Store Kit at 23-31.

“The SKPaymentTransaction class defines objects residing in the payment queue. A payment transaction is created whenever a payment is added to the payment queue. Transactions are delivered to your application when the App Store has finished processing the payment. Completed transactions provide a receipt and transaction identifier that your application can use to save a permanent record of the processed payment.”

**Source:** Store Kit at 32; *see also* Store Kit at 32-37.

“The SKReceiptRefreshRequest class allows an app to refresh its receipt. With this API, the app can request a new receipt if the receipt is invalid or missing. In the sandbox environment, you can request a receipt with any combination of properties to test the state transitions related to Volume Purchase Plan receipts.”

**Source:** Store Kit at 47; *see also* Store Kit at 47-49.

“The SKPaymentTransactionObserver protocol declares methods that are implemented by observers of an SKPaymentQueue object.

An observer is called when transactions are updated by the queue or removed from the queue. An observer should process all successful transactions, unlock the functionality purchased by the user, and then finish the transaction by calling the payment queue’s finishTransaction: (page 27) method.”

**Source:** Store Kit at 57; *see also* Store Kit at 57-61.

### “Downloading Hosted Content from Apple’s Server

**Note:** For information about associating Apple-hosted content with a product, see “Creating Products in iTunes Connect” (page 8).

When the user purchases a product that has associated hosted content, the

transaction passed to your transaction queue observer also includes an instance of SKDownload which lets you download the content.

To download the hosted content, add the download objects from the transaction's downloads property to the transaction queue by calling the startDownloads: method of SKPaymentQueue. If the value of the downloads property is nil, there is no hosted content for that transaction. Unlike downloading apps, downloading hosted content does not automatically require a Wifi connection for content larger than a certain size. Avoid using cellular networks to download large files without an explicit action from user.

Implement the paymentQueue:updatedDownloads: method on the transaction queue observer to respond to changes in a download's state—for example, by updating progress in your UI or by notifying the user of a failed download.

Update your user interface while the content is downloading using the values of the progress and timeRemaining properties. You can use the pauseDownloads:, resumeDownloads:, and cancelDownloads: methods of SKPaymentQueue from your UI to let the user control in-progress downloads. Use the downloadState property to determine whether the download has completed. Don't use the progress or time remaining, these properties wouldn't let your app accurately tell the difference between downloads that are almost finished and downloads that are completely finished.

After the download finishes, use its contentURL property to locate the content.”

**Source:** In-App Purchases 2013 at 28.

#### **“Downloading Content from Your Own Server**

As with all other interactions between your app and your server, the details and mechanics of this process are up to you. The communication consists of, at minimum, the following steps:

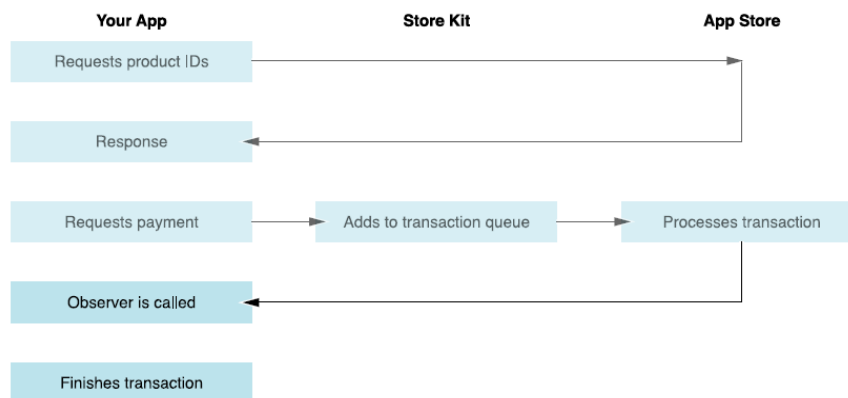
1. Your app sends the receipt to your server and requests the content.
2. Your server validates the receipt to establish that the content has been purchased, as described in Receipt Validation Programming Guide.
3. Assuming the receipt is valid, your server responds to your app with the content.

Consider the security implications of how you host your content and of how your app communicates with your server. For more information, see Security Overview.”

**Source:** In-App Purchases 2013 at 29.

In the last part of the purchase process, after the App Store has processed the payment request, your app stores information about the purchase for future launches, downloads the purchased content, and marks the transaction as finished, as shown in Figure 4-1 (page 21).

**Figure 4-1** Stages of the purchase process—delivering products

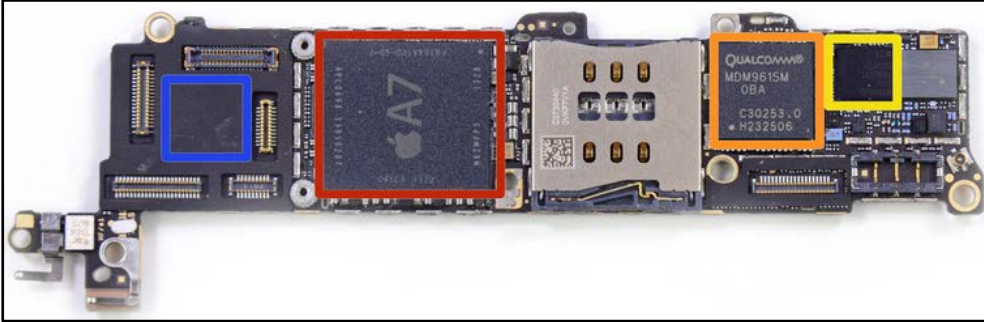


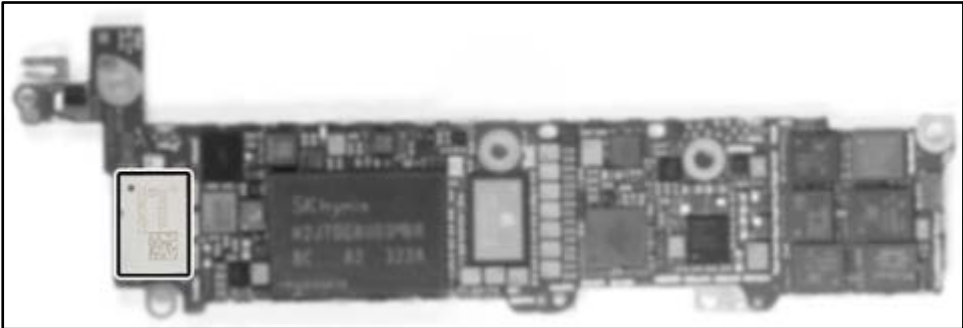
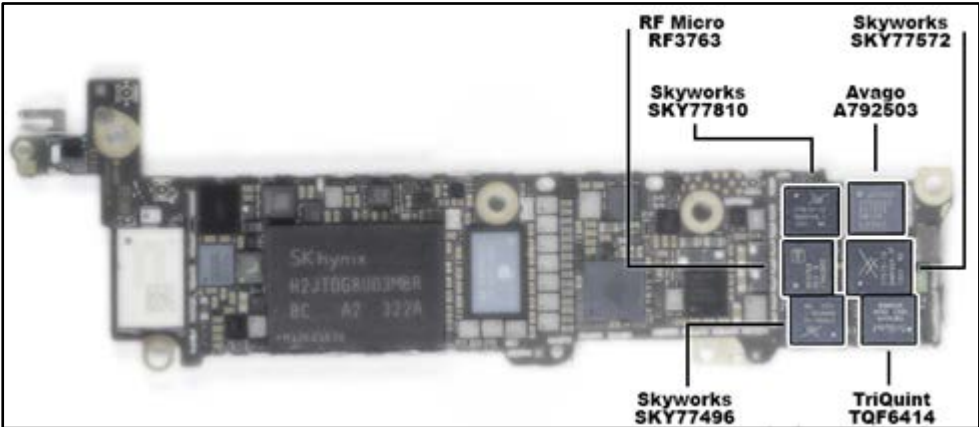
**Source:** In-App Purchases 2013 at 21 (showing transaction completion and content delivery).

Claim Element	Accused Instrumentalities
1a. A data access terminal for retrieving data from a data supplier and providing the retrieved data to a data carrier, the terminal comprising:	<p>Apple has committed and continues to commit acts of infringement under 35 U.S.C. § 271 (i) with any version of iTunes that can access the iTunes Store; (ii) with any version of the App Store app; (iii) with any version of any Apple hardware or software product (e.g., Apple’s various iPhone products, Apple’s various iPad products, Apple’s various Apple TV products, Apple’s various Mac computer products, Apple’s various operating system software, etc.) that includes any version of iTunes or the App Store app that can access the iTunes Store; (iv) with any version of the Mac App Store; (v) with any Apple hardware or software product that includes any version of the Mac App Store; and (vi) with Apple’s internal servers involved in operating Apple’s iTunes Store, Apple’s Mac App Store, Apple’s in-application payment functionality, and Apple’s iAd Network (collectively referred to as “Apple’s Accused Instrumentalities”).</p> <p>Robot Entertainment has committed and continues to commit acts of infringement under 35 U.S.C. § 271 with its “Hero Academy” app (when installed on an Apple device, such as an Apple iPhone).</p> <p>KingsIsle has committed and continues to commit acts of infringement under 35 U.S.C. § 271 with its “Grub Guardian” and “WizardBlox” apps (when</p>

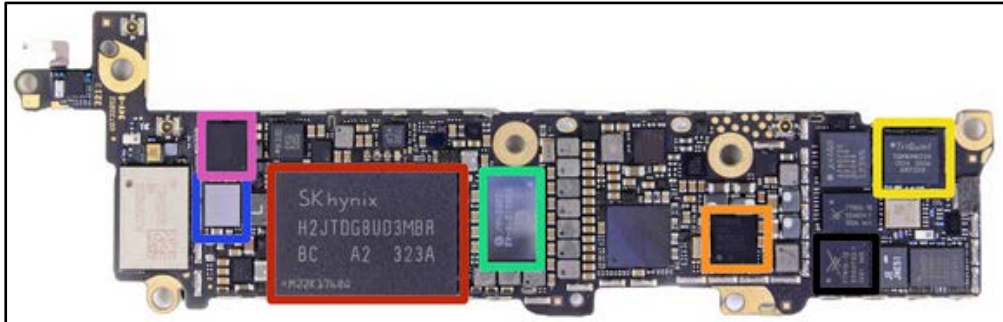
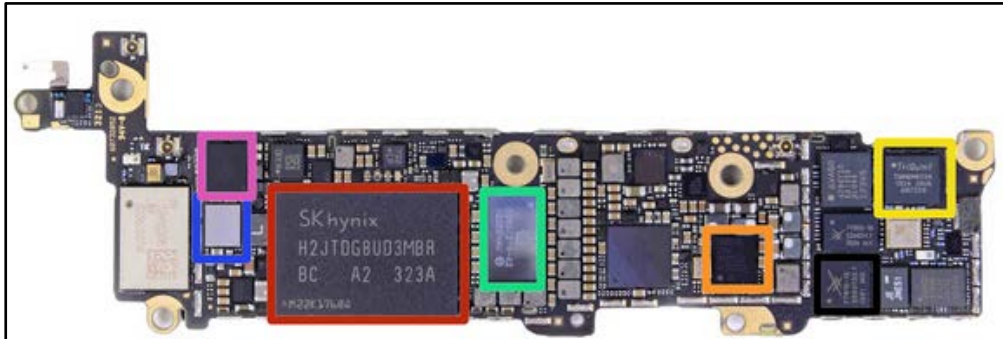
Claim Element	Accused Instrumentalities
	<p>installed on an Apple device, such as an Apple iPhone).</p> <p>Game Circus has committed and continues to commit acts of infringement under 35 U.S.C. § 271 with its apps (when installed on an Apple device, such as an Apple iPhone) that require payment and with its apps that use Apple's in-application payment functionality to collect payment for enhanced functionality or additional content and with its apps that provide in-application advertising.</p> <p>Apple's Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise an infringing data access terminal for retrieving data from a data supplier and providing the retrieved data to a data carrier.<sup>2</sup> See, e.g.,:</p>
1b. a first interface for communicating with the data supplier;	<p>Apple's Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise a first interface for communicating with the data supplier. See, e.g.,:</p> <p><b>“Ultrafast LTE wireless.</b> More LTE coverage than ever before.</p> <p>iPhone 5s has up to 13 LTE bands. That's more than any other single model of smartphone. Which means even more iPhone users can experience fast download and upload speeds in more places around the world. Meanwhile, the number of LTE carriers supported by iPhone worldwide continues to grow. So</p>

<sup>2</sup> See Apple, Inc., *Apple – iPhone 5s – Built-in Apps* (<http://www.apple.com/iphone-5s/built-in-apps/>) [hereinafter “Built-in Apps”]; see Apple, Inc., *Apple – iPhone 5s – Features* (<http://www.apple.com/iphone-5s/features/>) [hereinafter “Features”]; see Apple, Inc., *Apple – iPhone 5s – Technical Specifications* (<http://www.apple.com/iphone-5s/specs/>) [hereinafter “Specifications”]; Apple, Inc., *Hero Academy for iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPod touch (3rd generation), iPod touch (4th generation), iPod touch (5th generation) and iPad on the iTunes App Store* (<https://itunes.apple.com/us/app/hero-academy/id488156323?mt=8>) [hereinafter “Hero Academy iTunes”]; Apple, Inc., *Grub Guardian for iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPod touch (3rd generation), iPod touch (4th generation), iPod touch (5th generation) and iPad on the iTunes App Store* (<https://itunes.apple.com/us/app/grub-guardian/id498330099?mt=8>) [hereinafter “Grub Guardian iTunes”]; Apple, Inc., *Prize Claw for iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPod touch (3rd generation), iPod touch (4th generation), iPod touch (5th generation) and iPad on the iTunes App Store* (<https://itunes.apple.com/us/app/prize-claw/id434800417?mt=8&ign-mpt=uo%3D2>) [hereinafter “Prize Claw iTunes”]; iFixit, *iPhone 5s Teardown – Page 2 – iFixit* (<http://www.ifixit.com/Teardown/iPhone+5s+Teardown/17383/2>) [hereinafter “Teardown Page 2”]; Chipworks, Inc., *Inside the iPhone 5s* (<http://www.chipworks.com/en/technical-competitive-analysis/resources/blog/inside-the-iphone-5s/>) [hereinafter “Inside iPhone 5s”]; Apple, Inc., *iPhone User Guide for iOS 7 Software* (2013) [hereinafter “iPhone User Guide”]; Apple, Inc., *In-App Purchase Programming Guide* (Sep. 2012) [hereinafter “In-App Purchases 2012”]; Apple, Inc., *Store Kit Framework Reference* (Sept. 2013) [hereinafter “Store Kit”]; Apple, Inc., *In-App Purchase Programming Guide* (Sep. 2013) [hereinafter “In-App Purchases 2013”]; Apple, Inc., *Getting Started with In-App Purchase on iOS and OS X* (Sep. 2013) [hereinafter “Getting Started”]; Apple, Inc., *iTunes Store – Terms and Conditions*, ([www.apple.com/legal/internet-services/itunes/us/terms.html](http://www.apple.com/legal/internet-services/itunes/us/terms.html)) [hereinafter “iTunes Terms”]; Apple, Inc., *iTunes Connect Developer Guide* (Aug. 2013) [hereinafter “Connect”].

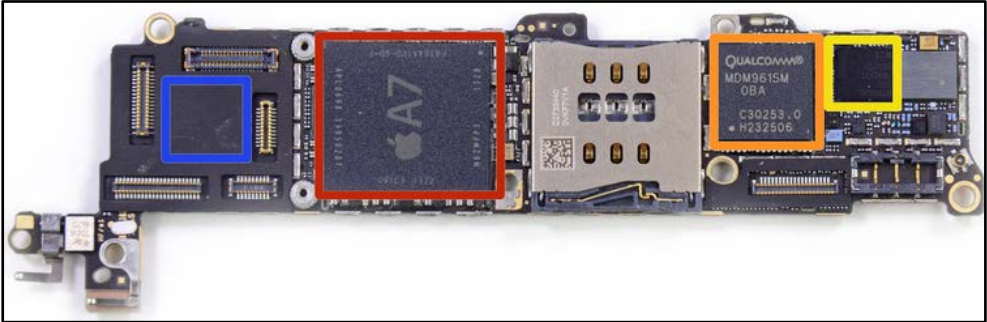
Claim Element	Accused Instrumentalities
	<p>when you're traveling, you can take advantage of ultrafast LTE networks in more places."</p> <p><b>Source:</b> Features.</p> <p><b>"Model A1533 (GSM)"</b> : UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz); LTE (Bands 1, 2, 3, 4, 5, 8, 13, 17, 19, 20, 25)</p> <p><b>Model A1533 (CDMA)"</b> : CDMA EV-DO Rev. A and Rev. B (800, 1700/2100, 1900, 2100 MHz); UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz); LTE (Bands 1, 2, 3, 4, 5, 8, 13, 17, 19, 20, 25)</p> <p><b>Model A1453"</b> : CDMA EV-DO Rev. A and Rev. B (800, 1700/2100, 1900, 2100 MHz); UMTS/HSPA+/DC-HSDPA (850, 900, 1700/2100, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz); LTE (Bands 1, 2, 3, 4, 5, 8, 13, 17, 18, 19, 20, 25, 26)</p> <p><b>Model A1457"</b> : UMTS/HSPA+/DC-HSDPA (850, 900, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz); LTE (Bands 1, 2, 3, 5, 7, 8, 20)</p> <p><b>Model A1530"</b> : UMTS/HSPA+/DC-HSDPA (850, 900, 1900, 2100 MHz); GSM/EDGE (850, 900, 1800, 1900 MHz); FDD-LTE (Bands 1, 2, 3, 5, 7, 8, 20); TD-LTE (Bands 38, 39, 40)</p> <p>802.11a/b/g/n Wi-Fi (802.11n 2.4GHz and 5GHz) Bluetooth 4.0 wireless technology"</p> <p><b>Source:</b> Specifications.</p>  <p><b>Source:</b> Teardown Page 2 (The component outlined in orange is the Qualcomm MDM9615M 4G LTE modem; the component outlined in yellow is the Qualcomm WTR1605L</p>

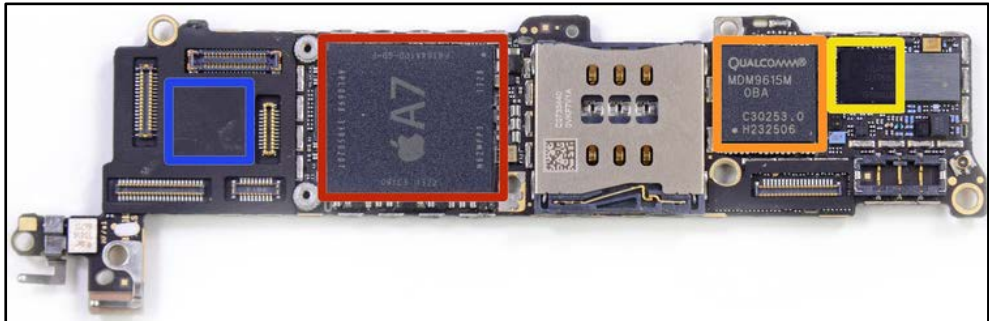
Claim Element	Accused Instrumentalities								
	<p>LTE/HSPA+/CDMA2K/TDSCDMA/EDGE/GPS transceiver).</p> <div></div> <p><b>Source:</b> Inside iPhone 5s (The highlighted module on the left contains the Broadcom BCM4332 chip, which has an integrated IEEE 802.11 a/b/g and single-stream IEEE 802.11n MAC/baseband/radio, Bluetooth 4.0, and FM radio receiver).</p> <div></div> <p><b>Source:</b> Inside iPhone 5s (The highlighted components on the right are the RF Micro RF3763 Power Amplifier-Duplexer (PAD) for B5 / 8 Dual PAD; Skyworks SKY77572 Band 18/19/20 Power Amplifier Device; Skyworks SKY77810 2G/EDGE Power Amplifier Module; Avago A792503 Band 25/3 Power Amplifier Device; Skyworks SKY77496 Band 13/17 Power Amplifier Device; and TriQuint TQF6414 Band 1/4 Dual Power Amplifier Device).</p>								
1c. a data carrier interface for interfacing with the data carrier;	<p>Apple’s Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise a data carrier interface for interfacing with the data carrier. <i>See, e.g.,</i>:</p> <table><tr><td>“Capacity and Price</td><td>16GB</td><td>32GB</td><td>64GB</td></tr><tr><td></td><td>\$199</td><td>\$299</td><td>\$399”</td></tr></table> <p><b>Source:</b> Specifications.</p>	“Capacity and Price	16GB	32GB	64GB		\$199	\$299	\$399”
“Capacity and Price	16GB	32GB	64GB						
	\$199	\$299	\$399”						

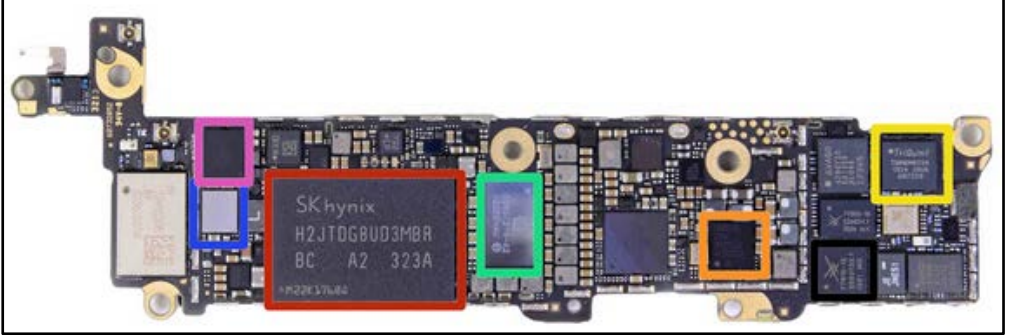


Claim Element	Accused Instrumentalities								
	<div></div> <p><b>Source:</b> Teardown Page 2 (The component outlined in red is the SK Hynix H2JTDG8UD3MBR 128 Gb (16 GB) NAND flash).</p>								
1d. a program store storing code implementable by a processor; and	<p>Apple’s Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise a program store storing code implementable by a processor. <i>See, e.g.,</i>:</p> <table><tr><td>“Capacity and Price</td><td>16GB</td><td>32GB</td><td>64GB</td></tr><tr><td></td><td>\$199</td><td>\$299</td><td>\$399”</td></tr></table> <p><b>Source:</b> Specifications.</p> <div></div> <p><b>Source:</b> Teardown Page 2 (The component outlined in red is the SK Hynix H2JTDG8UD3MBR 128 Gb (16 GB) NAND flash).</p> <p>“<b>A7 chip.</b> The first 64-bit smartphone in the world.</p> <p>There’s fast. And then there’s A7 fast. The new A7 chip gives you CPU and graphics performance up to 2x faster than the A6 chip. Even more impressive, A7 makes iPhone 5s the first 64-bit smartphone in the world — that’s desktop-class architecture in a superslim phone. And because iOS 7 was built specifically for 64-bit, it’s uniquely designed to take advantage of the A7 chip. A7 supports OpenGL ES version 3.0 to deliver the kind of detailed graphics and complex visual effects once possible only on Mac computers, PCs, and</p>	“Capacity and Price	16GB	32GB	64GB		\$199	\$299	\$399”
“Capacity and Price	16GB	32GB	64GB						
	\$199	\$299	\$399”						



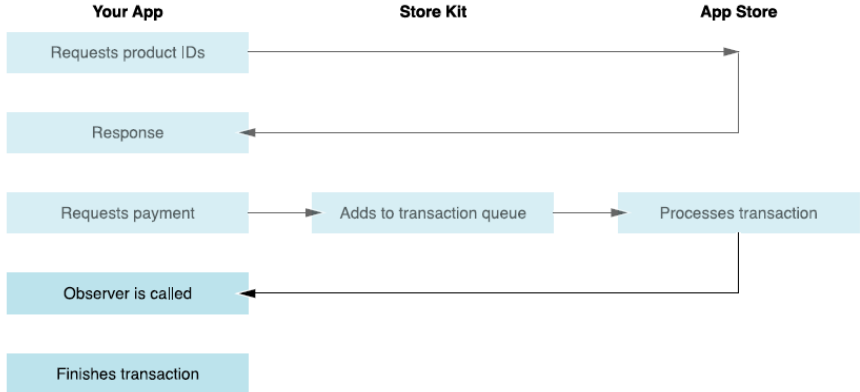
Claim Element	Accused Instrumentalities
	<p>gaming consoles. The difference is amazing. Take the imaginary worlds in games, for instance. Textures and shadows look more true to life. Sunlight reflects off the water. The whole experience feels much more realistic. iOS 7 and all the built-in apps are optimized for the A7 chip. The Camera app is a great example. It takes advantage of a new image signal processor built into A7 to give you up to 2x faster autofocus, faster photo capture, and higher video frame rates. You'd think with all this going on, battery life would suffer. But it doesn't, because A7 is designed to be incredibly energy efficient.”</p> <p><b>Source:</b> Features.</p> <p>“iOS 7 comes with an amazing collection of apps for the things you want to do and need to do, like sending email and texts, browsing the web, shooting and sharing great photos and videos, getting directions, and more. And all the built-in apps are optimized to take advantage of the A7 chip's high-performance 64-bit architecture.”</p> <p><b>Source:</b> Features.</p>  <p><b>Source:</b> Teardown Page 2 (The component outlined in red is the Apple A7 processor).</p>
<p>1e. a processor, coupled to the first interface, to the data carrier interface and to the program store for implementing the stored code, the code comprising:</p>	<p>Apple's Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise a processor, coupled to the first interface, to the data carrier interface and to the program store for implementing the stored code. <i>See, e.g.,</i>:</p> <p><b>“A7 chip.</b> The first 64-bit smartphone in the world.</p> <p>There's fast. And then there's A7 fast. The new A7 chip gives you CPU and graphics performance up to 2x faster than the A6 chip. Even more impressive, A7 makes iPhone 5s the first 64-bit smartphone in the world — that's desktop-class architecture in a superslim phone. And because iOS 7 was built</p>

Claim Element	Accused Instrumentalities								
	<p>specifically for 64-bit, it’s uniquely designed to take advantage of the A7 chip. A7 supports OpenGL ES version 3.0 to deliver the kind of detailed graphics and complex visual effects once possible only on Mac computers, PCs, and gaming consoles. The difference is amazing. Take the imaginary worlds in games, for instance. Textures and shadows look more true to life. Sunlight reflects off the water. The whole experience feels much more realistic. iOS 7 and all the built-in apps are optimized for the A7 chip. The Camera app is a great example. It takes advantage of a new image signal processor built into A7 to give you up to 2x faster autofocus, faster photo capture, and higher video frame rates. You’d think with all this going on, battery life would suffer. But it doesn’t, because A7 is designed to be incredibly energy efficient.”</p> <p><b>Source:</b> Features.</p> <p>“iOS 7 comes with an amazing collection of apps for the things you want to do and need to do, like sending email and texts, browsing the web, shooting and sharing great photos and videos, getting directions, and more. And all the built-in apps are optimized to take advantage of the A7 chip’s high-performance 64-bit architecture.”</p> <p><b>Source:</b> Features.</p> <div></div> <p><b>Source:</b> Teardown Page 2 (The component outlined in red is the Apple A7 processor).</p> <table><tr><td><b>“Capacity and Price</b></td><td>16GB</td><td>32GB</td><td>64GB</td></tr><tr><td></td><td>\$199</td><td>\$299</td><td>\$399”</td></tr></table> <p><b>Source:</b> Specifications.</p>	<b>“Capacity and Price</b>	16GB	32GB	64GB		\$199	\$299	\$399”
<b>“Capacity and Price</b>	16GB	32GB	64GB						
	\$199	\$299	\$399”						

Claim Element	Accused Instrumentalities
	 <p><b>Source:</b> Teardown Page 2 (The component outlined in red is the SK Hynix H2JTDG8UD3MBR 128 Gb (16 GB) NAND flash).</p>
<p>If. code to read payment data from the data carrier and to forward the payment data to a payment validation system;</p>	<p>Apple's Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise code to read payment data from the data carrier and to forward the payment data to a payment validation system. <i>See, e.g.,</i>:</p> <p>“The SKMutablePayment class defines a request to the Apple App Store to process payment for additional functionality offered by your application. A payment encapsulates a string that identifies a particular product and the quantity of that item the user would like to purchase.</p> <p>When a mutable payment is added to the payment queue, the payment queue copies the contents into an immutable request before queueing the request. Your application can safely change the contents of the mutable payment object.”</p> <p><b>Source:</b> Store Kit at 14; <i>see also</i> Store Kit at 14-16.</p> <p>“The SKPayment class defines a request to the Apple App Store to process payment for additional functionality offered by your application. A payment encapsulates a string that identifies a particular product and the quantity of those items the user would like to purchase.”</p> <p><b>Source:</b> Store Kit at 17; <i>see also</i> Store Kit at 17-21.</p> <p>“The SKPaymentQueue class provides a queue of payment transactions to be processed by the App Store. The payment queue communicates with the App Store and presents a user interface so that the user can authorize payment. The contents of the queue are persistent between launches of your app.</p>

Claim Element	Accused Instrumentalities
	<p>To process a payment, first attach at least one observer object to the queue. Then, add a payment object for the item the user wants to purchase. Each time you add a payment object, the queue creates a transaction object to process that payment and enqueues it to be processed. After payment is fulfilled, the queue updates the transaction object and then calls any observer objects to provide them the updated transaction. Your observer should process the transaction and then remove it from the queue.”</p> <p><b>Source:</b> Store Kit at 22; <i>see also</i> Store Kit at 23-31.</p> <div data-bbox="509 632 1515 1178" data-label="Diagram"> <p>In the second part of the purchase process, after the user has chosen to purchase a particular product, your app submits payment request to the App Store, as shown in Figure 3-1 (page 17).</p> <p><b>Figure 3-1 Stages of the purchase process—requesting payment</b></p> <pre> graph LR     subgraph Your_App [Your App]         A[Requests product IDs]         B[Response]         C[Requests payment]         D[Observer is called]         E[Finishes transaction]     end     subgraph Store_Kit [Store Kit]         F[Adds to transaction queue]     end     subgraph App_Store [App Store]         G[Processes transaction]     end     A --&gt; G     G --&gt; B     B --&gt; A     C --&gt; F     F --&gt; G     G --&gt; D     D --&gt; E   </pre> </div> <p><b>Source:</b> In-App Purchases 2013 at 17 (showing payment submission to App Store).</p>
<p>1g. code to receive payment validation data from the payment validation system;</p>	<p>Apple’s Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise code to receive payment validation data from the payment validation system. <i>See, e.g.,</i>:</p> <p>“The SKPaymentQueue class provides a queue of payment transactions to be processed by the App Store. The payment queue communicates with the App Store and presents a user interface so that the user can authorize payment. The contents of the queue are persistent between launches of your app.</p> <p>To process a payment, first attach at least one observer object to the queue. Then, add a payment object for the item the user wants to purchase. Each time you add a payment object, the queue creates a transaction object to process that payment and enqueues it to be processed. After payment is fulfilled, the</p>

Claim Element	Accused Instrumentalities
	<p>queue updates the transaction object and then calls any observer objects to provide them the updated transaction. Your observer should process the transaction and then remove it from the queue.</p> <p>The exact mechanism you use to process a processed transaction depends on the design of your app and the product being purchased. Here are a few common examples:</p> <ul style="list-style-type: none"> <li>● If the product is a feature already built into your app, then to process the transaction, your app would enable the feature.</li> <li>● If the product includes downloadable content provided by the App Store, your app would retrieve the SKDownload objects from the transaction and ask the payment queue to download them. You would provide the actual content files to be served by the App Store to iTunes Connect when you create the product information.</li> <li>● If the product represents downloadable content provided by your own server, your app might open a network connection to your server and download the content from there.”</li> </ul> <p><b>Source:</b> Store Kit at 22; <i>see also</i> Store Kit at 23-31.</p> <p>“The SKPaymentTransaction class defines objects residing in the payment queue. A payment transaction is created whenever a payment is added to the payment queue. Transactions are delivered to your application when the App Store has finished processing the payment. Completed transactions provide a receipt and transaction identifier that your application can use to save a permanent record of the processed payment.”</p> <p><b>Source:</b> Store Kit at 32; <i>see also</i> Store Kit at 32-37.</p> <p>“The SKReceiptRefreshRequest class allows an app to refresh its receipt. With this API, the app can request a new receipt if the receipt is invalid or missing. In the sandbox environment, you can request a receipt with any combination of properties to test the state transitions related to Volume Purchase Plan receipts.”</p> <p><b>Source:</b> Store Kit at 47; <i>see also</i> Store Kit at 47-49.</p> <p>“The SKPaymentTransactionObserver protocol declares methods that are implemented by observers of an SKPaymentQueue object.</p> <p>An observer is called when transactions are updated by the queue or removed from the queue. An observer should process all successful transactions, unlock</p>

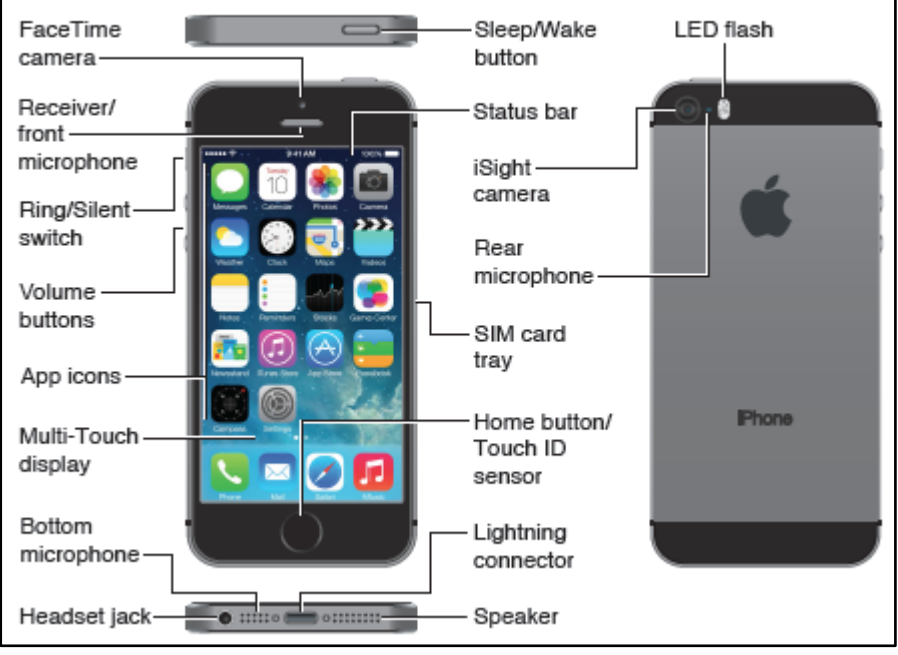
Claim Element	Accused Instrumentalities
	<p>the functionality purchased by the user, and then finish the transaction by calling the payment queue's finishTransaction: (page 27) method.”</p> <p><b>Source:</b> Store Kit at 57; <i>see also</i> Store Kit at 57-61.</p> <div data-bbox="509 415 1515 982" style="border: 1px solid black; padding: 10px;"> <p>In the last part of the purchase process, after the App Store has processed the payment request, your app stores information about the purchase for future launches, downloads the purchased content, and marks the transaction as finished, as shown in <a href="#">Figure 4-1</a> (page 21).</p> <p><b>Figure 4-1</b> Stages of the purchase process—delivering products</p>  <pre> sequenceDiagram     participant App as Your App     participant Kit as Store Kit     participant Store as App Store      App-&gt;&gt;Store: Requests product IDs     Store--&gt;&gt;App: Response     App-&gt;&gt;Kit: Requests payment     Kit-&gt;&gt;Store: Adds to transaction queue     Store-&gt;&gt;Kit: Processes transaction     Store--&gt;&gt;App: Observer is called     App-&gt;&gt;App: Finishes transaction           </pre> </div> <p><b>Source:</b> In-App Purchases 2013 at 21 (showing transaction completion and content delivery).</p>
<p>1h. code responsive to the payment validation data to retrieve data from the data supplier and to write the retrieved data into the data carrier.</p>	<p>Apple’s Accused Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple device, such as an Apple iPhone) comprise code responsive to the payment validation data to retrieve data from the data supplier and to write the retrieved data into the data carrier. <i>See, e.g.,</i>:</p> <p>“The SKPaymentQueue class provides a queue of payment transactions to be processed by the App Store. The payment queue communicates with the App Store and presents a user interface so that the user can authorize payment. The contents of the queue are persistent between launches of your app.</p> <p>To process a payment, first attach at least one observer object to the queue. Then, add a payment object for the item the user wants to purchase. Each time you add a payment object, the queue creates a transaction object to process that payment and enqueues it to be processed. After payment is fulfilled, the queue updates the transaction object and then calls any observer objects to provide them the updated transaction. Your observer should process the transaction and then remove it from the queue.</p>

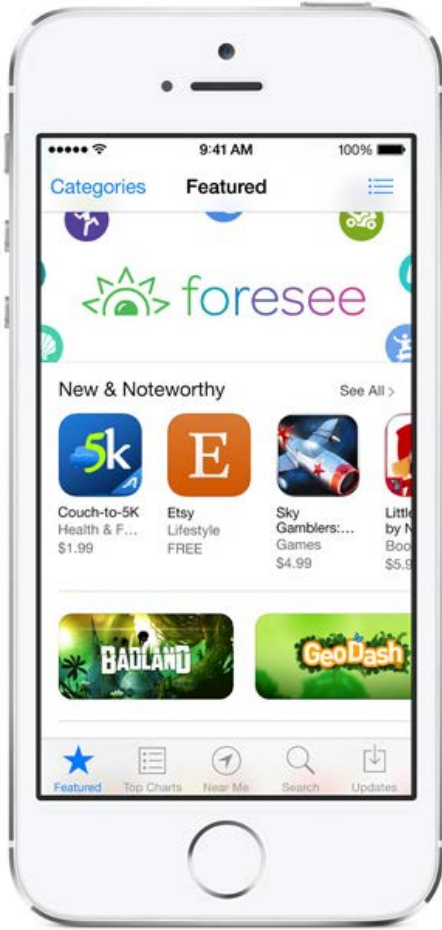

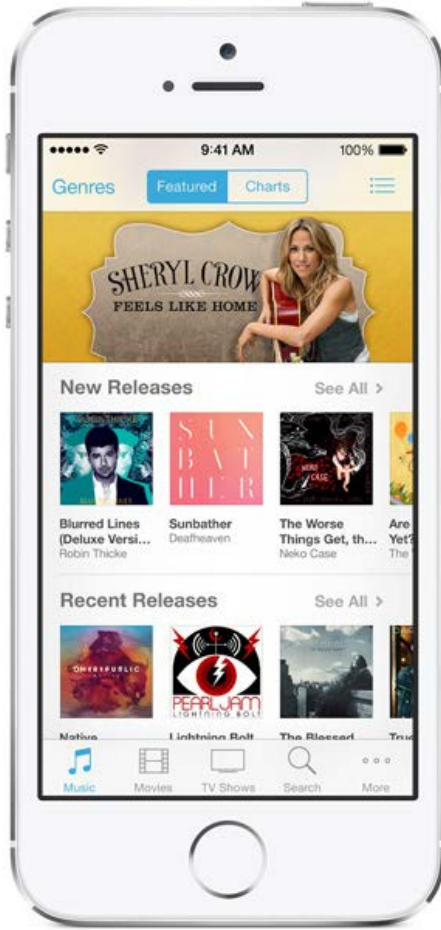

Claim Element	Accused Instrumentalities
	<p>The exact mechanism you use to process a processed transaction depends on the design of your app and the product being purchased. Here are a few common examples:</p> <ul style="list-style-type: none"> <li>● If the product is a feature already built into your app, then to process the transaction, your app would enable the feature.</li> <li>● If the product includes downloadable content provided by the App Store, your app would retrieve the SKDownload objects from the transaction and ask the payment queue to download them. You would provide the actual content files to be served by the App Store to iTunes Connect when you create the product information.</li> <li>● If the product represents downloadable content provided by your own server, your app might open a network connection to your server and download the content from there.”</li> </ul> <p><b>Source:</b> Store Kit at 22; <i>see also</i> Store Kit at 23-31.</p> <p>“The SKPaymentTransaction class defines objects residing in the payment queue. A payment transaction is created whenever a payment is added to the payment queue. Transactions are delivered to your application when the App Store has finished processing the payment. Completed transactions provide a receipt and transaction identifier that your application can use to save a permanent record of the processed payment.”</p> <p><b>Source:</b> Store Kit at 32; <i>see also</i> Store Kit at 32-37.</p> <p>“The SKReceiptRefreshRequest class allows an app to refresh its receipt. With this API, the app can request a new receipt if the receipt is invalid or missing. In the sandbox environment, you can request a receipt with any combination of properties to test the state transitions related to Volume Purchase Plan receipts.”</p> <p><b>Source:</b> Store Kit at 47; <i>see also</i> Store Kit at 47-49.</p> <p>“The SKPaymentTransactionObserver protocol declares methods that are implemented by observers of an SKPaymentQueue object.</p> <p>An observer is called when transactions are updated by the queue or removed from the queue. An observer should process all successful transactions, unlock the functionality purchased by the user, and then finish the transaction by calling the payment queue’s finishTransaction: (page 27) method.”</p> <p><b>Source:</b> Store Kit at 57; <i>see also</i> Store Kit at 57-61.</p>

Claim Element	Accused Instrumentalities
	<p><b>“Downloading Hosted Content from Apple’s Server</b></p> <p><b>Note:</b> For information about associating Apple-hosted content with a product, see “Creating Products in iTunes Connect” (page 8).</p> <p>When the user purchases a product that has associated hosted content, the transaction passed to your transaction queue observer also includes an instance of SKDownload which lets you download the content.</p> <p>To download the hosted content, add the download objects from the transaction’s downloads property to the transaction queue by calling the startDownloads: method of SKPaymentQueue. If the value of the downloads property is nil, there is no hosted content for that transaction. Unlike downloading apps, downloading hosted content does not automatically require a Wifi connection for content larger than a certain size. Avoid using cellular networks to download large files without an explicit action from user.</p> <p>Implement the paymentQueue:updatedDownloads: method on the transaction queue observer to respond to changes in a download’s state—for example, by updating progress in your UI or by notifying the user of a failed download.</p> <p>Update your user interface while the content is downloading using the values of the progress and timeRemaining properties. You can use the pauseDownloads:, resumeDownloads:, and cancelDownloads: methods of SKPaymentQueue from your UI to let the user control in-progress downloads. Use the downloadState property to determine whether the download has completed. Don’t use the progress or time remaining, these properties wouldn’t let your app accurately tell the difference between downloads that are almost finished and downloads that are completely finished.</p> <p>After the download finishes, use its contentURL property to locate the content.”</p> <p><b>Source:</b> In-App Purchases 2013 at 28.</p> <p><b>“Downloading Content from Your Own Server</b></p> <p>As with all other interactions between your app and your server, the details and mechanics of this process are up to you. The communication consists of, at minimum, the following steps:</p> <ol style="list-style-type: none"> <li>1. Your app sends the receipt to your server and requests the content.</li> <li>2. Your server validates the receipt to establish that the content has been purchased, as described in Receipt Validation Programming Guide.</li> </ol>



Claim Element	Accused Instrumentalities
	<p>3. Assuming the receipt is valid, your server responds to your app with the content.</p> <p>Consider the security implications of how you host your content and of how your app communicates with your server. For more information, see Security Overview.”</p> <p><b>Source:</b> In-App Purchases 2013 at 29.</p> <div data-bbox="509 598 1516 1167" data-label="Diagram"> <p>In the last part of the purchase process, after the App Store has processed the payment request, your app stores information about the purchase for future launches, downloads the purchased content, and marks the transaction as finished, as shown in Figure 4-1 (page 21).</p> <p><b>Figure 4-1 Stages of the purchase process—delivering products</b></p> <pre> graph LR     subgraph "Your App"         A[Requests product IDs]         B[Response]         C[Requests payment]         D[Observer is called]         E[Finishes transaction]     end     subgraph "Store Kit"         F[Adds to transaction queue]     end     subgraph "App Store"         G[Processes transaction]     end     A --&gt; G     G --&gt; B     C --&gt; F     F --&gt; G     G --&gt; D     D --&gt; E   </pre> <p>The diagram illustrates the stages of the purchase process. It shows three main components: Your App, Store Kit, and App Store. The process starts with the Your App requesting product IDs from the App Store. The App Store responds with a response. Then, the Your App requests payment, which is added to the transaction queue by the Store Kit. The Store Kit then processes the transaction with the App Store. Finally, the App Store calls the observer in the Your App, and the transaction is finished.</p> </div> <p><b>Source:</b> In-App Purchases 2013 at 21 (showing transaction completion and content delivery).</p>
<p>11. A data access terminal according to claim 1 integrated with at least one of a mobile communication device, a personal computer, an audio/video player, and a cable or satellite television interface device.</p>	<p>Apple’s Accused iOS Instrumentalities and the apps of Robot Entertainment, KingsIsle, and Game Circus (when installed on an Apple iOS device, such as an Apple iPhone) comprise an infringing data access terminal as claimed in claim 1, see above, wherein said data access terminal is integrated with at least one of a mobile communication device, a personal computer, an audio/video player, and a cable or satellite television interface device. <i>See, e.g.,</i>:</p>

Claim Element	Accused Instrumentalities
	<p><b>iPhone 5s</b></p>  <p>The diagram illustrates the components of an iPhone 5s from three perspectives: front, back, and side. Labels with leader lines point to the following parts:</p> <ul style="list-style-type: none"> <li><b>Front View Labels:</b> FaceTime camera, Receiver/front microphone, Ring/Silent switch, Volume buttons, App icons, Multi-Touch display, Bottom microphone, Headset jack.</li> <li><b>Back View Labels:</b> LED flash, iSight camera, Rear microphone, SIM card tray, Home button/Touch ID sensor, Lightning connector, Speaker.</li> <li><b>Side View Label:</b> Sleep/Wake button.</li> </ul> <p><b>Source:</b> iPhone User Guide at 8.</p>

Claim Element	Accused Instrumentalities
	<div data-bbox="537 233 976 1157">  </div> <div data-bbox="570 1199 878 1293">  App Store         </div> <div data-bbox="565 1304 997 1520"> <p>The best collection of mobile apps is just a tap away. The App Store has over 900,000 apps — many of them free — to help you be more productive.<sup>5</sup> Learn about the world. And let's not forget, play games.</p> </div> <div data-bbox="565 1535 909 1604"> <p><a href="#">Learn more about apps from the App Store &gt;</a></p> </div> <div data-bbox="500 1625 802 1667"> <p><b>Source:</b> Built-in Apps.</p> </div> <div data-bbox="1036 233 1474 1157">  </div> <div data-bbox="1065 1199 1419 1293">  iTunes Store         </div> <div data-bbox="1060 1304 1505 1520"> <p>With millions of songs and thousands of TV shows and HD movies, it's easy to find your favorites — and discover new ones — when you shop the iTunes Store. With iCloud, you can access everything you buy on all your devices.</p> </div> <div data-bbox="1060 1535 1347 1566"> <p><a href="#">Learn more about iCloud &gt;</a></p> </div>